**OBJETOS PROGRAMABLES BASE DE DATOS I**

**Fecha de entrega** [25/06/2025]

**Elaborado por:** Daniel Sierra

**Carrera:** Ing. Software

**Dirigido a:** Ana María Pazmiño D

**OBJETIVO PROPUESTO DE ESTUDIO:**

Demostrar dominio sobre el conocimiento de objetos programables en SQL **Ejercicios de desarrollo:**

|  |
| --- |
| **Enunciado:** |
| Sobre la tabla Orders Details de la base de datos Northwind, se requiere realizar un objeto programable que valide la cantidad de registros a eliminar, este número no debe exceder a 1 por cada eliminación |
| **Argumentación:** |
| se utiliza un trigger porque es el mecanismo adecuado para controlar y validar las operaciones que se realizan directamente sobre una tabla en la base de datos, de forma automática y transparente para el usuario. |
| **Scripts:** |
| CREATE TRIGGER ValidacionEliminacion  ON [Order Details]  AFTER DELETE  AS  BEGIN  IF (SELECT COUNT (\*) FROM deleted) >1  BEGIN  RAISERROR('NO SE PUEDE ELIMINAR UN RESGISTRO MAS DE UNA VEZ',16,  1)  ROLLBACK  END  ELSE  BEGIN  PRINT 'La eliminacion se hizo'  END  END |
| **Resultado por error:** |
|  |
| **Resultado por acierto:** |
|  |
|  |
|  |

|  |
| --- |
| **Enunciado:** |
| Se requiere hacer un objeto programable que permita el reporte de las ventas totales por empleados |
| **Argumentación:** |
| Se utiliza una vista porque permite encapsular y reutilizar la consulta que calcula las ventas totales por empleado, facilitando el acceso a la información sin necesidad de reescribir el código. Además, mejora la legibilidad, seguridad y mantenimiento del sistema al brindar una forma estructurada y consistente de consultar datos resumidos. |
| **Scripts:** |
| USE Northwind  GO  --Se requiere hacer un objeto programable que permita el reporte de las ventas totales por empleados  SELECT E.FirstName +' '+ E.LastName AS NombreEmpleado,  ROUND (SUM((OD.Quantity \* OD.UnitPrice)/(1+OD.Discount)),2) AS VentaTotal  FROM Orders O  INNER JOIN Employees E ON O.EmployeeID = E.EmployeeID  INNER JOIN [Order Details] OD ON O.OrderID = OD.OrderID  GROUP BY E.FirstName, E.LastName  CREATE VIEW VW\_VtaEmpleadoTotal  AS  SELECT E.FirstName +' '+ E.LastName AS NombreEmpleado,  ROUND (SUM((OD.Quantity \* OD.UnitPrice)/(1+OD.Discount)),2) AS VentaTotal  FROM Orders O  INNER JOIN Employees E ON O.EmployeeID = E.EmployeeID  INNER JOIN [Order Details] OD ON O.OrderID = OD.OrderID  GROUP BY E.FirstName, E.LastName |
| **Resultado por error:** |
| *[Captura la o las imágenes que del resultado del procedimiento realizado.]* |
| **Resultado por acierto:** |
| SELECT \* FROM VW\_VtaEmpleadoTotal  ORDER BY VentaTotal DESC |
|  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Enunciado:** | | Se requiere hacer un objeto programable que permita el acceso a parámetros de rango de fechas para realizar un reporte de ventas totales por empleado | | **Argumentación:** | | Se utiliza un procedimiento almacenado porque permite ejecutar una consulta parametrizada para generar reportes dinámicos según un rango de fechas definido por el usuario. Esto mejora la flexibilidad, reutilización y seguridad delacceso a los datos, al mismo tiempo que permite validar las fechas antes de ejecutar la consulta. | | **Scripts:** | | --Se requiere hacer un objeto programable que permita el acceso a parámetros de rango de fechas para realizar un reporte de ventas totales por empleado  CREATE PROCEDURE SP\_Q\_VtaEmpleadoTotalFechas  @FechaInicio as VARCHAR(10),  @FechaFin VARCHAR(10)  AS  BEGIN  IF @FechaInicio > @FechaFin  BEGIN  PRINT'La fecha inicio no puede ser mayor a la fecha fin, velva a intentar'  END  ELSE  BEGIN  SELECT E.FirstName +' '+ E.LastName AS NombreEmpleado,  O.OrderDate,  ROUND (SUM((OD.Quantity \* OD.UnitPrice)/(1+OD.Discount)),2) AS VentaTotal  FROM Orders O  INNER JOIN Employees E ON O.EmployeeID = E.EmployeeID  INNER JOIN [Order Details] OD ON O.OrderID = OD.OrderID  WHERE O.OrderDate BETWEEN CONVERT(datetime, @FechaInicio, 101) AND CONVERT(datetime, @FechaFin, 101)  GROUP BY E.FirstName, E.LastName, O.OrderDate  END  END | | **Resultado por error:** | |  | | **Resultado por acierto:** |  |  | | --- | | **Scripts:** | | USE Northwind  GO  --Se requiere hacer un objeto programable que permita el reporte de las ventas totales por empleados  SELECT E.FirstName +' '+ E.LastName AS NombreEmpleado,  ROUND (SUM((OD.Quantity \* OD.UnitPrice)/(1+OD.Discount)),2) AS VentaTotal  FROM Orders O  INNER JOIN Employees E ON O.EmployeeID = E.EmployeeID  INNER JOIN [Order Details] OD ON O.OrderID = OD.OrderID  GROUP BY E.FirstName, E.LastName  CREATE VIEW VW\_VtaEmpleadoTotal  AS  SELECT E.FirstName +' '+ E.LastName AS NombreEmpleado,  ROUND (SUM((OD.Quantity \* OD.UnitPrice)/(1+OD.Discount)),2) AS VentaTotal  FROM Orders O  INNER JOIN Employees E ON O.EmployeeID = E.EmployeeID  INNER JOIN [Order Details] OD ON O.OrderID = OD.OrderID  GROUP BY E.FirstName, E.LastName | | **Resultado por error:** | | *[Captura la o las imágenes que del resultado del procedimiento realizado.]* | | **Resultado por acierto:** | | SELECT \* FROM VW\_VtaEmpleadoTotal  ORDER BY VentaTotal DESC | |  |  |  | | --- | | *[Captura la o las imágenes que del resultado del procedimiento realizado.]* | | *SELECT de corroboración:*  *[Realiza un select antes de llamar al procedimiento y uno después de realizar el procedimiento.]* | |  |   **Enunciado:** |
| Se requiere hacer un objeto programable que permita el ingreso de ventas  (order details) |
| **Argumentación:** |
| Se utiliza un procedimiento almacenado para gestionar el ingreso de ventas porque permite centralizar la lógica de validación, controlar errores, garantizar la integridad de los datos y facilitar su reutilización desde diferentes aplicaciones o interfaces. Además, mejora la seguridad y el mantenimiento del sistema al evitar inserciones directas en la tabla. |
| **Scripts:** |

|  |
| --- |
| --Se requiere hacer un objeto programable que permita el ingreso de ventas (order detail)  CREATE PROCEDURE SP\_I\_DetalleVentas  @OrderID INT,  @ProductID INT,  @UnitPrice MONEY,  @Quantity SMALLINT,  @Discount REAL  AS  BEGIN  BEGIN TRY  -- Iniciar la transacción  BEGIN TRANSACTION;  -- Validaciones básicas  IF @OrderID IS NULL OR @ProductID IS NULL  BEGIN  RAISERROR('OrderID y ProductID no pueden ser NULL.', 16, 1);  ROLLBACK TRANSACTION;  RETURN;  END  IF @UnitPrice < 0 OR @Quantity <= 0  BEGIN  RAISERROR('UnitPrice debe ser >= 0 y Quantity debe ser > 0.', 16, 1);  ROLLBACK TRANSACTION;  RETURN;  END  IF @Discount < 0 OR @Discount > 1  BEGIN  RAISERROR('El descuento debe estar entre 0 y 1.', 16, 1);  ROLLBACK TRANSACTION;  RETURN;  END  -- Validar existencia de la orden y el producto  IF NOT EXISTS (SELECT 1 FROM Orders WHERE OrderID = @OrderID)  BEGIN  RAISERROR('No existe una orden con el OrderID proporcionado.', 16, 1);  ROLLBACK TRANSACTION;  RETURN;  END  IF NOT EXISTS (SELECT 1 FROM Products WHERE ProductID = @ProductID)  BEGIN  RAISERROR('No existe un producto con el ProductID proporcionado.', 16, 1);  ROLLBACK TRANSACTION;  RETURN;  END  -- Insertar el detalle de venta  INSERT INTO [Order Details] (OrderID, ProductID, UnitPrice, Quantity, Discount)  VALUES (@OrderID, @ProductID, @UnitPrice, @Quantity, @Discount);  -- Confirmar la transacción  COMMIT TRANSACTION;  END TRY  BEGIN CATCH  -- En caso de error, deshacer transacción y mostrar error  IF @@TRANCOUNT > 0  ROLLBACK TRANSACTION;  DECLARE @ErrorMessage NVARCHAR(4000), @ErrorSeverity INT, @ErrorState INT;  SELECT  @ErrorMessage = ERROR\_MESSAGE(),  @ErrorSeverity = ERROR\_SEVERITY(),  @ErrorState = ERROR\_STATE();  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);  END CATCH  END |
| **Resultado por error:** |
|  |
| **Resultado por acierto:** |
|  |
| *SELECT de corroboración:* |
|  |